Report Number: WUCS-93-6

1993-01-01

SYMPHONY: A Hardware, Operating System, and Protocol Processing Architecture for Distributed Multimedia Applications

Andreas D. Bovopoulos, R. Gopalakrishnan, and Saied Hosseini

This paper explores the architectural requirements for computers to be able to process multimedia data streams such as video and audio. The I/O subsystem is shown to be a bottleneck, and a network backplane approach is suggested to alleviate this. The need to provide end-to-end performance guarantees requires predictable performance of intra-machine communication, and a schedulable bus with reservation is proposed to achieve this. In addition this requires operating system (OS) mechanisms to negotiate and enforce QoS requirements of applications. A real-time microkernel executive is proposed for each autonomous unit. Requirements for real-time microkernel exeutive is proposed for each... **Read complete abstract on page 2.**

SYMPHONY: A Hardware, Operating System, and Protocol Processing Architecture for Distributed Multimedia Applications

Andreas D. Bovopoulos, R. Gopalakrishnan, and Saied Hosseini

Complete Abstract:

This paper explores the architectural requirements for computers to be able to process multimedia data streams such as video and audio. The I/O subsystem is shown to be a bottleneck, and a network backplane approach is suggested to alleviate this. The need to provide end-to-end performance guarantees requires predictable performance of intra-machine communication, and a schedulable bus with reservation is proposed to achieve this. In addition this requires operating system (OS) mechanisms to negotiate and enforce QoS requirements of applications. A real-time microkernel executive is proposed for each autonomous unit. Requirements for real-time microkernel exeutive is proposed for each autonomous unit. Requirements for real-time scheduling and efficient interprocess communication mechanisms are described. Finally the implications of the hardware and OS enhancements for the protocol procesing mechanisms are discussed. A compositional approach to protocol organization that is capable of catering to the wide variations in transport requirements of various media is described.

SYMPHONY: A Hardware, Operating System, and Protocol Processing Architecture for Distributed Multimedia Applications

Andreas D. Bovopoulos, R. Gopalakrishnan and
Saied Hosseini

WUCS-93-06

March 1993

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130-4899

This work was supported by an industrial consortium of Ascom Timeplex, Bellcore, BNR, DEC, Gold Star Information and Communications, Italtel SIT, NEC America, NTT and SynOptics Communications.

SYMPHONY: A Hardware, Operating System, And Protocol Processing Architecture For Distributed Multimedia Applications

Andreas D. Bovopoulos[*] *R. Gopalakrishnan*[*] *Saied Hosseini*[#]

[*]Department of Computer Science [#]Department of Electrical Engineering
and
Computer and Communications Research Center
Washington University in Saint Louis, USA

Abstract

This paper explores the architectural requirements for computers to be able to process multimedia data streams such as video and audio. The I/O subsystem is shown to be a bottleneck, and a network backplane approach is suggested to alleviate this. The need to provide end-to-end performance guarantees requires predictable performance of intra-machine communication, and a schedulable bus with reservations is proposed to achieve this. In addition this requires operating system (OS) mechanisms to negotiate and enforce QoS requirements of applications. A real-time microkernel executive is proposed for each autonomous unit. Requirements for real-time scheduling and efficient interprocess communication mechanisms are described. Finally the implications of the hardware and OS enhancements for the protocol processing mechanisms are discussed. A compositional approach to protocol organization that is capable of catering to the wide variations in transport requirements of various media is described.

1. Introduction

Distributed processing of multimedia data is becoming a widespread reality, mainly due to the following factors: (i) cost effective and high performance VLSI for microprocessors, DSP, and coding/compression hardware, (ii) advances in broadband networking, (iii) advancement in fiber-optic transmission technology, (iv) market pressure for new service deployment. Advances in high speed networking are being made possible by the development of the Asynchronous Transfer Mode (ATM) technology, which will provide performance guarantees that make it possible to support connections whose parameters (such as bandwidth, maximum delay, delay jitter and call types) can vary dynamically and assume wide range of values. For an ATM network to support multimedia applications, performance guarantees must be provided not only within the ATM network but on an end-to-end basis, i.e. at the application level. The goal of current research into distributed multimedia systems is to extend QoS guarantees available at the edge of the network up to the end-applications within the end-system. It is becoming increasingly evident, however, that this goal cannot be reached without fundamental changes to the architecture, operat-

ing system and applications architecture of the end-systems. The present project aims to address this problem by proposing *SYMPHONY: a hardware, operating system, and protocol processing architecture for distributed multimedia applications.*

The proposed SYMPHONY organization differs significantly from current implementations as follows. The SYMPHONY hardware architecture provides direct paths from the network interface to data sources/inks with predictable and tailored transfer characteristics, a scalable I/O bus architecture, and back-to-back cell processing at high input link rates of up to 620 Mbps. The SYMPHONY operating system organization incorporates the notion of QoS in its resource management mechanisms, which are reflected in its resource models, scheduling policies and resource sharing mechanisms. The SYMPHONY communication software organization caters to the wide diversity in media types, connection modes, and application requirements that will arise out of the capability of end-systems and networks to handle multimedia data.

2. Design Goals of SYMPHONY

With the computer becoming more of a tool for interactive communication, designing the communication and computing components in isolation of one another is inadequate and inappropriate for offering multimedia services. The SYMPHONY architecture is based on the proposition that in order to provide multimedia services with performance guarantees effectively, an architecture must be integrated with respect to the following three aspects:

- Hardware design (*Architecture Integration*),
- Protocol support (*Service Integration*),
- QoS specification (*Performance Integration*).

Given the above requirements, the design goals for the SYMPHONY architecture are formulated as:

1. Translate the high transmission rates provided by the network into application level throughput.
2. Develop an architectural framework for integrating subsystems such as processors, storage, and digital I/O devices for different media. Furthermore, define hardware and software interfaces to interconnect subsystems.
3. Allow negotiation of performance parameters and provide guaranteed performance of the various subsystems that are used by an application.
4. Provide protocol support for communicating and processing digital streams of different media types with media and application specific performance requirements.

In the following subsections, the above design goals will dictate design choices affecting the SYMPHONY hardware, operating systems and communications software organization. The advantages of the proposed architecture when compared with more traditional architectures will be demonstrated.

3. Hardware Organization of SYMPHONY

In the previous section four design goals for the SYMPHONY architecture were identified. The first three of the design goals affect the SYMPHONY hardware organization.

1. Translating the high transmission rates available on network links into application level throughput. To achieve this, a network I/O interface capable of transferring cells at negotiated rates, to hardware units attached to this interface at a negotiated rate is required. A viable strategy for providing this is to use separate paths for unrelated data streams, and to eliminate multiplexing and asynchronous sharing of

physical data paths for in-band data. An alternate mechanism is to provide a multiplexed I/O interconnect that can guarantee the negotiated rate between the respective devices.

2. Developing an architectural framework for integrating subsystems, and defining hardware and software interfaces to interconnect subsystems. To achieve this a uniform design of each I/O interface that can be dimensioned or parameterized for each processing unit depending on the unit's capacity is required.

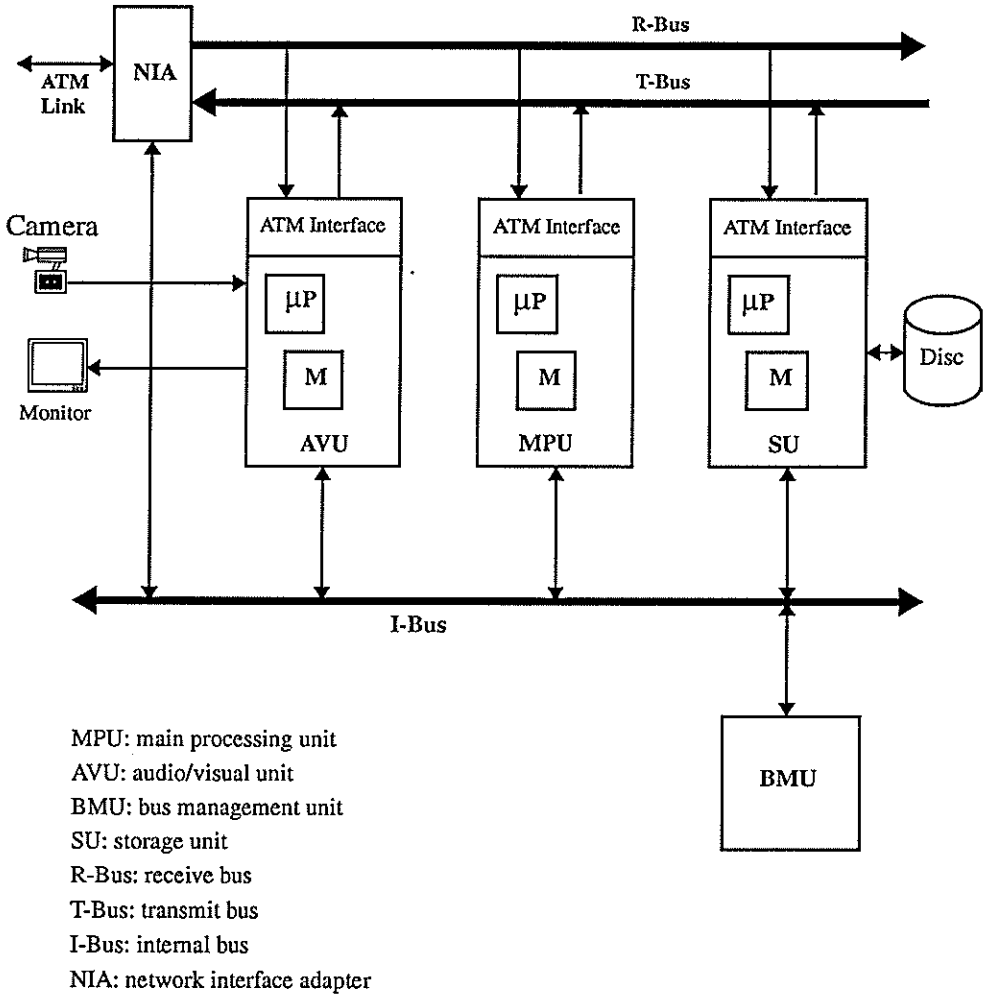


Figure 1: SYMPHONY Hardware Organization

3. Allowing negotiation of performance parameters and providing guaranteed performance of the various subsystems that are used by an application. To achieve this, mechanisms are required at the hardware level to enforce the negotiated performance requirements between each application and the Operating System (OS). It's important that the negotiation be provided along the entire processing path for ensuring end-to-end performance. For example, there could be a file transfer application, that streams File Transfer Protocol Data Units (FTPDUs) directly from the disk controller unit to the network I/O unit. Thus, the OS must ensure sufficient bandwidth between the disk controller and the network I/O unit to meet the negotiated transfer rate. However a more conventional application would read FTPDUs into

the application buffers over an internal bus and so in addition, bandwidth must be reserved on the internal bus as well.

This above requirements have lead to the consideration of a number of choices for the hardware organization of SYMPHONY. The chosen design is depicted in Figure 1 and is based on a systems integration approach which has the advantages of openness, simplicity of prototyping and satisfies the requirements listed above. The hardware organization of SYMPHONY is viewed as a collection of specialized, self-contained, autonomous units that communicate and coordinate their actions to execute an application. These units are typically the main processing unit (MPU), the storage unit (SU), the network interface adaptor (NIA), a graphics display controller, and multimedia boards for video and audio. The local resources and activities of each peripheral unit are managed by a locally executing kernel that serves requests from other units and can be controlled by the MPU. It is assumed that the quality of service provided by each unit is negotiable and guaranteed by it's local kernel.

A crucial aspect of the SYMPHONY hardware organization is the I/O interconnection architecture. In the rest of this subsection, the design of two major components of the interconnection mechanism -namely the ATM network I/O backplane and the system I/O backplane, are discussed in detail. In addition, the design of the arbitration mechanism used and the bus interfaces for each device on these backplanes is presented.

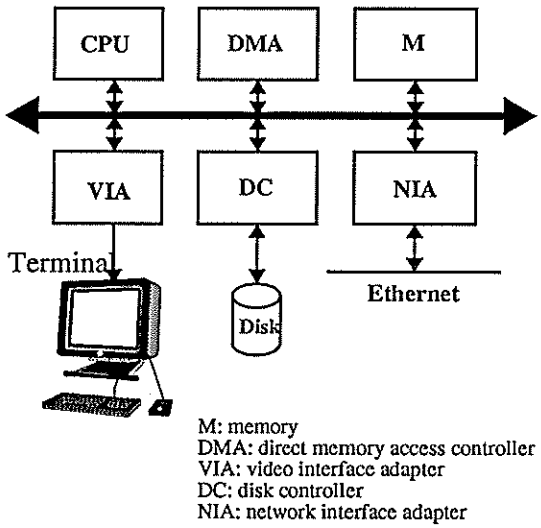


Figure 2: Conventional Hardware Organization

3.1. The Network I/O Backplane

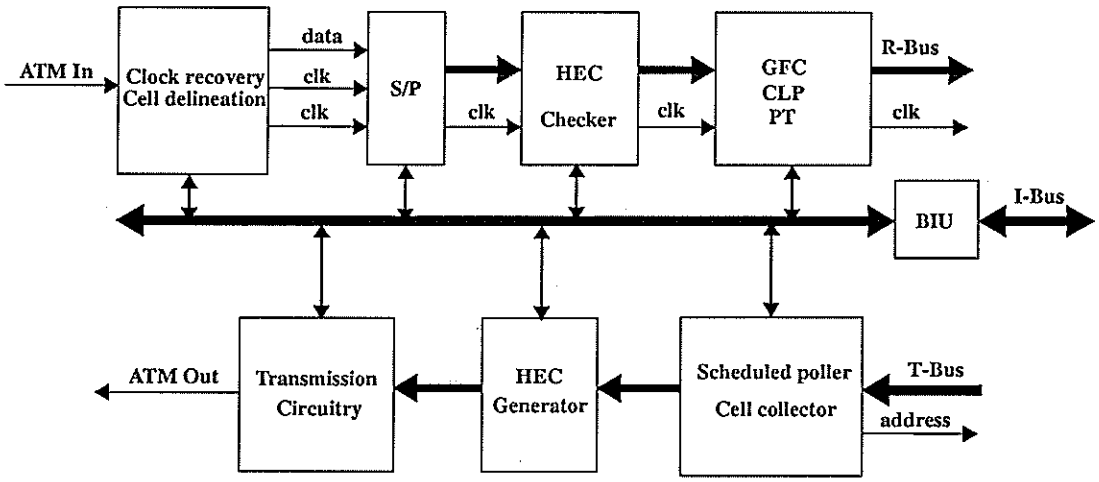
This section describes the design of the ATM network interface and its interaction with the rest of the system. In order to understand the factors that motivated the design, it is necessary to review how network connectivity is provided in a conventional workstation.

In a conventional bus based workstation (see Figure 2), the network interface adaptor (NIA) is attached to the system bus and is accessed as an I/O device by the main CPU. Typically, data to be transmitted is copied into a linked list of frames located in shared dual ported memory. The CPU then issues a command to the NIA, which then reads the common memory and transmits the frame according to the specific network

protocol. Similarly, incoming data is stored in the Receive Frame area in shared memory, and the system CPU is informed of data arrival by an interrupt from the NIA.

This organization of the network interface is unsuitable for several reasons. First, the load on the common system bus can be heavy and unpredictable. This introduces a substantial and variable amount of delay between the generation of a frame and its transmission. It is therefore difficult to assure throughput and delay bounds for network traffic. Another drawback that is especially problematic for multimedia traffic such as video, is that large volumes of network traffic load the main I/O bus and degrade system performance as a whole[35]. Thus unhindered data transfer over critical paths must be provided, so as to be able to provide assured bounds on delay and data transfer rate.

In order to solve the above problems, the *network backplane* approach shall be used. The network backplane provides a separate physical access path to the network interface for each device in SYMPHONY. It also supports multicast within the machine in a natural manner. The arbitration mechanism that coordinates access to the backplane assures negotiated performance and can be implemented entirely in hardware. Also the design is scalable as link speeds increase and new media types are introduced. The backplane has two data buses and a set of control lines that are controlled by the Network Interface Adaptor. These components are described below.



GFC: generic flow control
CLP: cell loss priority
PT: payload type check
HEC: header error correction
BIU: bus interface unit

Figure 3: The NIA Unit

3.1.1. The NIA unit

The network interface adaptor, as the name suggests, is physically attached to the network medium. It handles the low level hardware functions such as clock recovery, signal encoding/decoding, and serial/parallel conversion. The NIA terminates two buses - the R-Bus and the T-Bus. Once a cell is received over the network, the NIA places it on the R-Bus to be read by the downstream devices. It also collects cells from each upstream device over the T-Bus and multiplexes them over the ATM network channel. The NIA is the only bus master over the T-Bus and R-Bus, and thus there is no overhead of bus arbitration or contention.

The logic on the NIA ensures that bus bandwidth is allocated in proportion to the bandwidth demand of each device on the bus. The NIA also supports local broadcast by providing a loopback mechanism.

3.1.2. The R-Bus

The R-Bus originates from the NIA and downstream devices attach to the R-Bus through a standard interface. The R-Bus consists of data and control lines. The control lines mainly consist of the clock, framing and data valid signals. The data bus width depends on speed requirements and could be up to 32 bits wide. A device attached to the R-Bus samples the VCI and VPI fields of each cell sent from the NIA. If the connection with this VCI/VPI has an endpoint on the device, it reads in the rest of the cell and passes it on to the next stage. It is easy to see that this design supports multicast. In addition the design allows protocol processing for each connection to be performed on the destination device, thus bypassing the main CPU altogether. As will be seen later, this design has many implications for the organization of communications software.

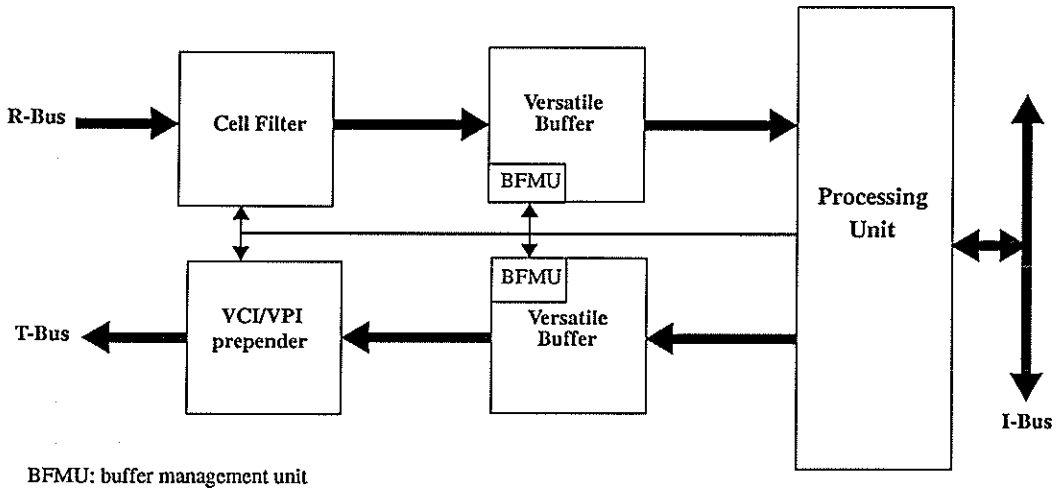


Figure 4: The ATM Interface

3.1.3. The T-Bus and its Scheduling Algorithm

The T-Bus terminates at and is controlled by the NIA. Upstream devices attach to it and communicate with the NIA through a cell addressable shared memory resident on each device. The memory on each device is viewed as an ATM cell queue that is written to by the device and read by the NIA. The NIA manages the T-Bus as a time multiplexed resource that is allocated to a device in proportion to the aggregate link bandwidth that has been reserved for connections that have endpoints on this device. The part of the NIA that controls the T-Bus uses a scheduling algorithm to poll the devices based on the traffic description of each connection emanating from that device. The intent is to implement appropriate scheduling schemes in hardware. The T-Bus could have up to a 32 bit data path, control lines for accessing shared device memory, and status lines originating from each device.

A unique aspect of the T-Bus design, not present in conventional systems, is the scheduling mechanism to share the link bandwidth between connections that originate from the different devices on the system. This has been made possible because ATM provides a connection oriented network abstraction with nego-

tiated connection characteristics which is known to the NIA scheduler. The actual scheduling algorithm will take into consideration any traffic shaping and/or policing mechanism imposed by the network at the User Network Interface (UNI). The scheduler uses information about existing connections in order to decide if it can support a new connection with the requested parameters. Once a connection request is accepted the scheduler must ensure that the delay is bounded within the machine and that cells are drained at the negotiated rate. A number of alternative scheduling algorithms may be used, including static priority algorithms (such as rate monotonic scheduling) or dynamic priority algorithms based on deadlines [23].

3.2. The System I/O Backplane

This section describes how local communication between units is effected in SYMPHONY. It is as important to provide uniform and predictable performance for intra-machine operations as it is to provide such performance for inter-machine operations, since a local operation could very well be the bottleneck in the processing path of an application. However unlike network I/O traffic, the communication between devices is not connection oriented and involves variable data sizes. Nevertheless, certain applications such as high speed animation, video editing and high speed file transfers generate large volumes of data at almost fixed intervals and require internal data paths with an assured capacity. Therefore it is desirable to arbitrate usage of the system bus to deliver predictable performance. The hardware design of SYMPHONY incorporates this requirement in the I-Bus design, in order to achieve guaranteed performance by allowing devices on the bus to make resource reservations, as is described below.

3.2.1. Conventional Bus Operation

The conventional bus has data, address and control lines which form a backplane to which other devices are attached. The arbitration unit (or arbiter) enforces a protocol among devices in order to ensure correct bus operation. At any given time, the arbiter makes one of the devices the *bus master,* which makes exclusive use of the address and data lines. A device indicates its need to use the bus by raising a bus grant signal to the arbiter. Depending on the priority of the requester(s), the arbiter grants the bus to one requester, which then becomes the bus master.

The problem with this scheme is that, once a device becomes the master, it must release the bus (in most cases) voluntarily, before another device, is able to access the bus. Also, the priority mechanism is static because the bus grant signal is physically daisy chained from one device to the other in order to reduce the number of wires on the backplane. Therefore a device cannot expect to have access to the bus when it desires, and wide variations in the delay between the bus request and bus grant may result. In addition, the bus mechanisms are primarily meant to support infrequent and bursty usage patterns. Finally, most devices use caches and other mechanisms to reduce their traffic over the bus.

However, the data and the traffic patterns generated by multimedia devices have different characteristics, and hence different abstractions and mechanisms are required. First, the data is not reusable, and hence caching techniques cannot be used. Second, the data is of a fairly periodic nature, and the burst sizes are predictable. Therefore it is worthwhile to consider a resource reservation approach to provide such sources guaranteed response times and throughput for intra machine data transfer [1,35].

3.2.2. I-Bus: A Schedulable Bus with Reservations

In the SYMPHONY architecture, the functionality of the arbitration unit (BMU) is enhanced, so that a sending device can specify the interval(s) between bus requests and the duration of each active period. For

example, an application providing a video retrieval service would make reservations for retrieving video frames for one or more sessions, each of which could have different retrieval rates. The BMU operates in a deadline driven fashion in order to allocate the bus to each device according to the temporal pattern of its requests. For the period during which no scheduled request is active, the bus is asynchronously shared by any other requesting devices in the normal fashion. The BMU will use algorithms to determine when a particular request pattern can be satisfied and a priority scheme to resolve conflicts. The algorithms used will be very similar to the ones used in the switching nodes within the network, but will be simpler because of negligible feedback delay between devices. The device drivers for each device must know the request patterns of the applications using the device and must communicate with the arbiter using a hardware protocol which will be developed as part of this project. This has implications for the applications and the system call interface provided by the operating system because they must be able to specify the request patterns as well as volume of data generated to exploit the facility provided by the bus interface.

4. Operating System Organization for SYMPHONY

The goal of providing mechanisms for an application to specify and negotiate performance imposes requirements on the operating system. This goal was motivated by the fact that applications have end-to-end performance requirements, and therefore the OS must be aware of application performance requirements and must provide mechanisms to enforce them[6,32]. There is a fundamental difference in the way the modeling and allocation of resources and the scheduling of activities is done within the network and within the machine. The network is modeled as a collection of communication resources (such as bandwidth, memory and processing power)[17]. This model allows the distributed control mechanism to allocate resources and schedule activities according to the QoS needs of the connections. On the other hand, the conventional time shared computer system is modeled as a *virtual machine* so that it appears as if each process has a *copy* of each resource for itself. A *copy* is mapped to its physical counterpart whenever the latter becomes available and no higher priority process is waiting for it. Once this mapping is set up, the processor is allocated to a process for a certain *time slice*. One difficulty in preserving QoS within the computer arises from the fact that a process has no control over the position or the duration of its time slice with respect to real time. Thus there is no attempt to coordinate related activities in time, which leads to unpredictable variations in the time taken to carry out periodic activities. Another difficulty stems from the fact that the resource scheduling is not in proportion to client requirements, but aims to satisfy unrestricted and unpredictable demands for resources from clients. This has resulted in the operating system evolving to provide a feature rich set of abstractions with little attention paid to providing predictable and negotiable performance. In the SYMPHONY environment the operating system in order to schedule different applications takes into consideration the requested resources and the required QoS.

The need for negotiability and predictability in operation will mainly affect policies for resource and task scheduling, resource modeling and management. It will also impact to some extent the system call interface, device driver interfaces and communications software support. An overview of suggested operating system organizations for multimedia support is presented and will form the basic context of our discussion of the above-mentioned issues.

4.1. Specialized Microkernel Executives

The need for flexibility, maintainability and extensibility drives current research in software systems[5,24,32]. This is especially true for operating system software for distributed and communication intensive environments. The role of the operating system is primarily to provide basic functionality for

resource management and interprocess communication (IPC). A key idea is the separation of policy and the mechanism used to implement it, which is effected by having a microkernel providing basic mechanisms to support abstractions for tasks, address spaces and IPC. The traditional operating system functions are implemented as user level processes and use the kernel primitives to implement various resource management policies. This approach is more modular and can be specialized for specific requirements.

The industry standard Mach© operating system microkernel will be used as the basic microkernel. Mach provides basic abstractions of tasks and threads, address spaces and communication ports. Initial prototyping of the OS support will be implemented over the NeXTSTEP© 3.0 object oriented operating system on a NeXT workstation with Objective-C support. Since the SYMPHONY architecture is composed of specialized units, each managing its resources autonomously, a natural choice would be to have a *specialized microkernel executive* running on each unit. User level tasks will be used to implement real time scheduling of active resources for each unit. A set of server processes executing on a unit allow local and remote tasks (on another unit or different machine) to access its resources. The servers are implemented as application objects, with well-defined procedural interfaces. The object invocation mechanism can be implemented with an efficient remote procedure call (RPC) mechanism. A common machine transparent interface must be provided to applications.

A user application is executed by one or more tasks on the MPU (see Figure 1). These tasks access other resources as clients, in accordance with the client-server paradigm. A client task may change the QoS of specific resources that it uses through that resource's service interface. Communication traffic between tasks on the machine can be optimized over the I-Bus. Resources and their servers are identified through a nameserver task running on the MPU.

4.2. Resource Management Policies

The resource management policies are critical with regard to providing predictable and timely performance. Resource management mainly involves resource modeling and allocation. Allocation is usually referred to as scheduling for active resources such as processors and the network and system backplanes. Passive resources, such as memory and disk blocks, must also be allocated, but will be assumed to be allocated locally.

4.2.1. Processor Scheduling

Processor scheduling is performed by the scheduler tasks executing on the processors of each specialized unit. The algorithms being considered are earliest-deadline-first, least-laxity-first and rate monotonic scheduling. Since most time critical events involve activities relating to real-time data transmission, reception and presentation, a static rate monotonic approach will be adequate and has the advantage of having a low overhead. The period of requests depends on the nature of the media involved. Furthermore, a varying amount of data may be generated while in the active state. The scheduler normally performs a schedulability test to make sure that a process's needs can be met. It then must generate a schedule that is feasible.

4.2.2. Network I/O Backplane Scheduling

When an application task wishes to communicate with a remote task, it creates a new network connection or becomes a new endpoint of an existing network connection [14]. An endpoint specifies the QoS it requires in terms of bandwidth and delay parameters. The connection manager task that runs on the NIA attempts to set up a connection with the requested characteristics and returns a success or failure notification to the requesting task. Apart from negotiating the QoS inside the network, the NIA must also ensure that the

required bandwidth and delay constraints are not violated within the NIA itself or the network I/O backplane. To do this the NIA performs a schedulability test to ensure that both active resources, such as the processor, and passive resources such as local memory, are sufficient to ensure connection quality. The NIA also generates a schedule for polling the send queues of devices that are on the T-bus. The polling mechanism can be implemented in hardware using a bus protocol. It would be desirable to take advantage of accurate traffic specifications to achieve a high utilization of the network backplane.

4.2.3. I-Bus Scheduling

The device managing this resource is the BMU and is accessible to other application and system tasks through an arbitration server task. This task is responsible for processing requests for bus usage from other tasks and using these requests to direct the arbitration mechanism. The server exports a procedural interface and is accessible as an application object. Asynchronous requests for bus usage are implemented in hardware for purposes of speed and efficiency.

4.2.4. Task Scheduling

Application programs in Mach are implemented as tasks containing one or more threads. Task management involves maintaining task state and the scheduling of threads. The scheduler can be implemented outside the kernel as a scheduler task and can be tuned to manage the activities on each unit Tasks are scheduled in a time shared manner. However, the deadlines associated with any periodic activity (such as generation of a video frame) are also taken into account. An important aspect of handling multimedia streams is synchronization of concurrent related streams in time[27,29,32,33,34,36]. This means that two or more tasks executing on separate processors must be scheduled according to some prespecified temporal relationship. The scheduler task provides a mechanism to do this through its interface functions.

4.3. Inter Task Communication Using The Object Based Communications Model

The Mach kernel implements the message passing IPC abstraction among task threads. It uses the notion of ports and port rights to address the threads within the tasks. The ports can be either local or remote, and port rights can be transferred between tasks in messages. This IPC mechanism can be used to implement network protocols such as TCP/IP at the level of user tasks. The NeXT operating system provides an object based interface for tasks built on top of the Mach message ports facility. It supports communication between local as well as remote objects, supports various parameter passing modes, and can handle objects as method arguments. The higher level interface provided by the object based communications model, allows the changing of application behavior dynamically in a structured manner. Since most system services are implemented as tasks, the object model makes it easy to build client-server systems with a higher level procedural interface. The object model is very useful for an incremental and compositional approach to building and experimenting with communication software.

5. SYMPHONY Communication Software Architecture

Supporting multimedia traffic with adequate performance is the primary goal for the SYMPHONY. The applications envisaged for the high speed packet switched networks have diverse and demanding transport requirements for which effective and well accepted solutions are not currently available. Distributed multimedia applications also require communication abstractions and associated primitives to model their

multipoint communication patterns and complex session control requirements. In this section, a communications software approach that directly addresses these two goals is presented. This software approach will be tightly coupled with SYMPHONY hardware and operating system organization, which provide the architectural framework on which the communication support will be built.

5.1. Decentralized Protocol Architecture

The design goals of SYMPHONY that significantly impact the communications software architecture are:

1. Provide hardware and OS support for real-time data traffic with QoS guarantees without degrading performance of local I/O.
2. Provide protocol support for a diverse mix of media, application requirements and connection modes.

The design strategy adopted in order to meet these objectives is:

1. Provide a separate path for network I/O and partition its bandwidth among the units in the machine.
2. Separate the incoming cell stream depending on the type of media it carries and deliver it to its destination device.

The design strategy described above is the rationale for the decentralized architecture which can be described informally as "perform the protocol processing where the data originates or is destined." The data from connections that comprise an application session are processed concurrently at their respective destination devices by media and application specific protocol modules. This implies a uniform and common execution environment for protocol modules on each unit, multiple independent state machines for an application's connections possibly maintained on different units, and mechanisms to set up and manage connections with specified QoS on the various units. Furthermore, the processing of data units belonging to the connections of an application must be coordinated in accordance with the dependencies that could exist between them. These dependencies could arise out of temporal relationships, causal relationships or user program intervention. Therefore the architecture must provide mechanisms to synchronize activities of related connections and to act upon a set of connections as a whole. This implies an abstraction for associating connections and referring to their controlling entities, as well as run time mechanisms to synchronize the protocol modules that maintain connection state. In subsequent sections, the manner in which the SYMPHONY design achieves these objectives is explored.

5.2. Execution Environment for Communication Modules in SYMPHONY

As described in the previous section, SYMPHONY distributed applications are implemented over the Mach operating system and are organized as application objects that communicate using method invocation. This kind of operating environment is currently provided by the NeXTSTEP 3.0, the object-oriented operating system on NeXT workstations. The initial implementation of the proposed communication software architecture will be done in this environment. The communication software architecture is implemented as a collection of system servers and user level tasks, each forming a component of the overall communications infrastructure. The distributed object model for applications provides a rich set of runtime support mechanisms which allow the mapping of the decentralized protocol processing architecture onto a collection of system and user level tasks cooperating via a high level procedural interface. Some of the advanced features of distributed objects, such as remote invocation, object arguments, and object transfer, will be used to implement the runtime mechanisms to setup and coordinate connections of an application that could be active on different devices. The granularity of an object instance is well suited for encapsulation of connec-

tion state and the resources associated with it, along with procedures for manipulating its properties during its lifetime. In addition, organization of protocol modules into a class hierarchy is a structured methodology for reducing complexity from the software engineering point of view. In this general architectural setting, the three main components of the communications software of SYMPHONY are now identified and described.

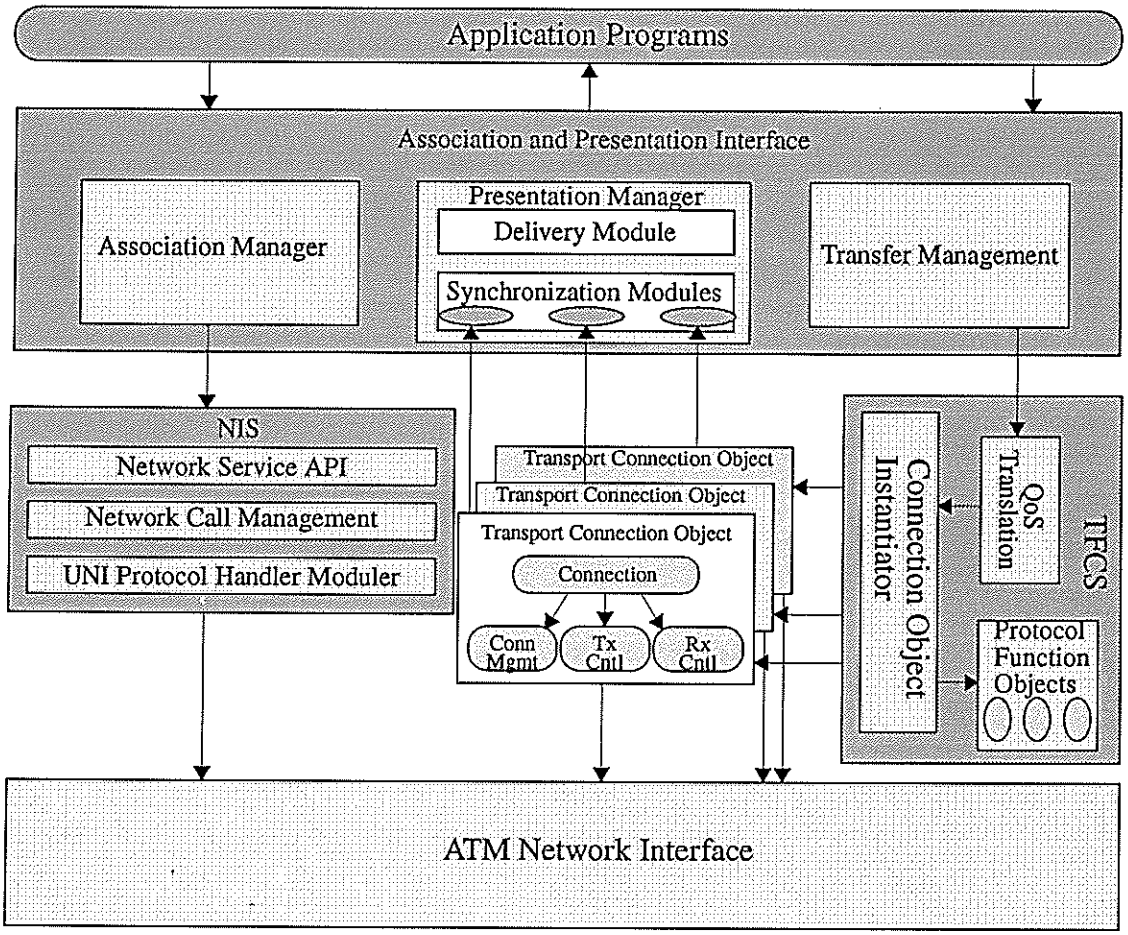


Figure 5: SYMPHONY Protocol Processing System Architecture

5.3. Network Interface System (NIS)

The NIS is responsible for handling client requests for network services. The NIS software resides on the NIA unit and is accessible through an interface provided by a server task running on the NIA. The main modules of NIS are the Application Programming Interface (API)[2,3,4] that provides a procedural interface to clients, the call manager that maintains the call and connection status for local clients, and a UNI protocol module that implements the signaling protocol between the host and the network. These three aspects of the NIS are elaborated upon below.

5.3.1. Network Service API

The API is a procedural interface that allows a user to gain access to network services. The procedural interface provides transparency from the signaling protocols that are exchanged across the User Network Interface (UNI). The network service interface that we use is the *call* model [7,14,18] that has been implemented for the Washington university campus ATM network[11]. The API functions allow a user to setup a call, supply participant addresses, and setup data connections with suitable transport characteristics. The attributes of the call and subsequent operations vary depending upon the call type used. The existing model for the call binds a set of addresses and allocates network resources among them. A call enfolds a set of *connections* with specific QoS parameters. Therefore the API has procedures to create and manipulate calls, add connections and negotiate their service parameters.The API will be implemented as a user level object that gets instantiated depending on the application request.

5.3.2. Call Manager

The call manager maintains state information on the calls that the local node participates in. As the network abstraction for ATM is connection oriented, it is necessary to maintain state information that includes the list of connections per call, the VCI/VPI to be used for transmitting on each connection, the VCIs/VPIs that have been assigned to other endpoints of the connections in the call, the transport characteristics per connection, and any other relevant information. The call manager also performs operations on a call in response to API functions invoked from above.

5.3.3. UNI Protocol Module

The ATM network is accessed through a signalling mechanism at the lowest level. The signaling occurs at the network edges known as the User Network Interface (UNI). The signaling protocol that will be used is called the Connection Management Access Protocol (CMAP) and is implemented for the Washington University campus network. Similar signaling standards are being developed by other bodies and can be incorporated as and when they are adopted.

5.4. The Transport Function Composition System (TFCS)

The TFCS is responsible for binding a set of transport function objects to a transport connection. The choice of algorithms for protocol functions is guided by the *transport service category* (TSC)[37] of the connection (see Figure 6), or can be explicitly specified by the creator during the establishment phase. The QoS requested is used to guide the choice of parameters (if any) with which to instantiate the objects chosen. These choices also involve negotiation with other endpoints of the connection. The main tasks of the TFCS are:

1. Matching TSCs to a set of protocol function objects (PFOs),
2. Negotiation of QoS with the network,
3. Instantiating the connection and the associated function objects with the negotiated parameters at the appropriate device.

5.4.1. Matching Transport Requirements to Mechanisms

This phase is guided by the TSC of the connection and the transport service quality requested. The TSC is specified by the application along with the QoS parameters.The TSC determines the choice of the set of objects that implement various protocol functions.The QoS is used to determine the parameters with which

these objects are instantiated. The choice of parameter values is determined by methods defined in the chosen objects. The chosen configuration is then bound to the connection. For example, consider an isochronous (i.e samples generated periodically) voice stream being multicast over a connection. The TSC would dictate a rate-based multicast transmission strategy, no acknowledgment generation and no error control.The parameters of relevance would be the size of a transmission unit (the number of bytes in a voice sample), the maximum transmission delay (150 msec), and the delivery rate to the recipient(s) (chosen to be the sampling interval at the source). The choices made above are then negotiated with the NIS as well as the remote peer, if the application requires it.

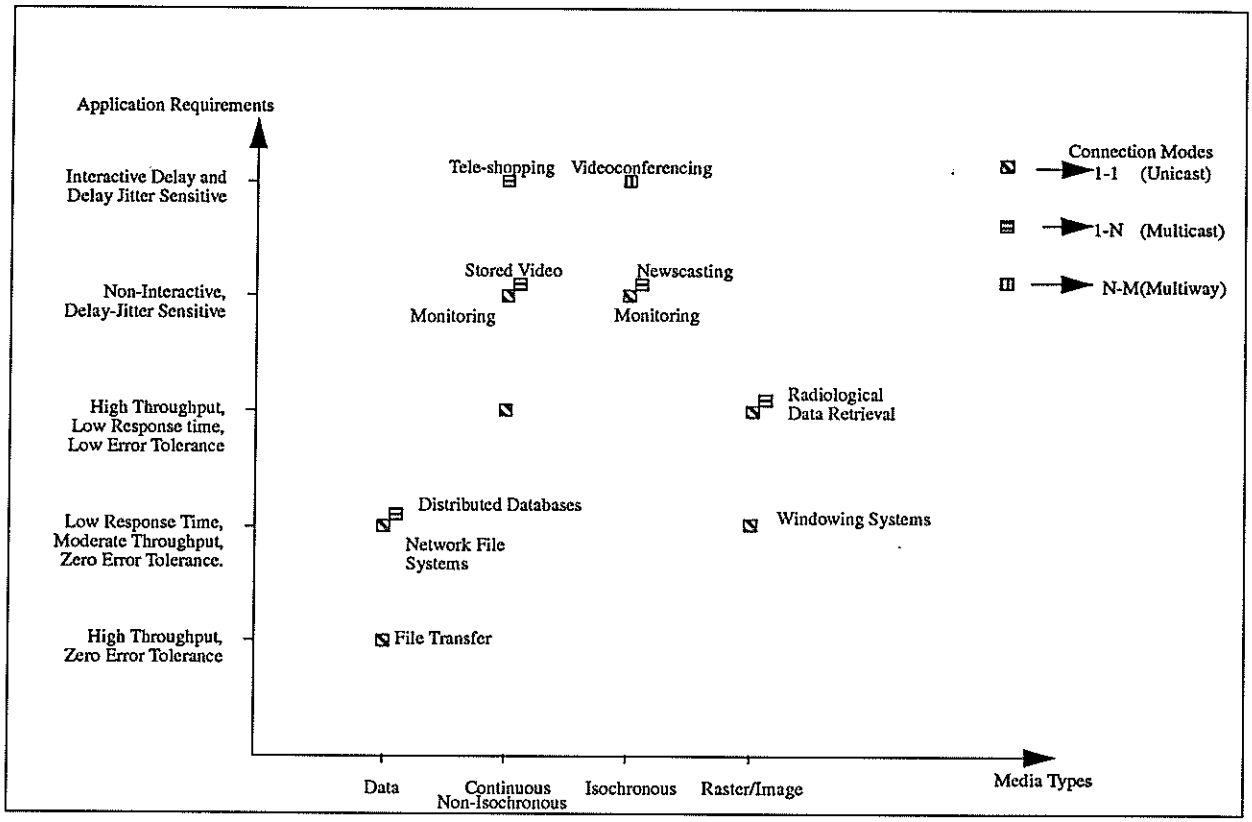


Figure 6: Transport Service Categories [33]

5.4.2. Negotiation with Service Provider

When an application wants to communicate, it must establish a call and add connections to it with requisite properties.This is done by invoking appropriate network services through the API mentioned above.The parameters to be passed are the set of endpoint addresses that participate in the call, as well as a list of connections and their transport characteristics. If network resources are unavailable, then the parameters are either relaxed or a failure status is reported.

The application may also require negotiation at the transport system level. This would be necessary in order to inform the other endpoint(s) of the specific transport algorithms that have been composed for the connection. This negotiation is done on a separate connection between the TFCS entities on each system.

5.4.3. Instantiating the Transport Function Objects

Once the transport protocol mechanisms have been agreed upon and the transport service parameters have been determined, the protocol objects must be instantiated, and associated to each other according to their functional relationships to perform protocol processing on data exchanged over the connection. These instantiated protocol objects composing the processing pipeline constitute a connection and are managed by a connection object. Each connection object is associated with a connection endpoint and resides on the processing unit where the endpoint is located. The connection object stores state information and pointers to the objects that support the connection. It also collects statistics and performs overall control of the transport service provided to the connection, such as reconfiguration of connection components.

5.4.4. Transport Function Class Hierarchy

Due to flexibility requirements and the necessity of aiding experimentation, the main transport layer functions have been identified and implemented as separate objects. The granularity of the objects from which a protocol suite is composed is much smaller than the layer by layer composition found in current systems. The objects comprising the transport functions library implement different algorithms for the general transport functions such as connection management, transmission control and reception control. Figure 6 shows the components of the general transport functions, which have been organized in a class hierarchy.

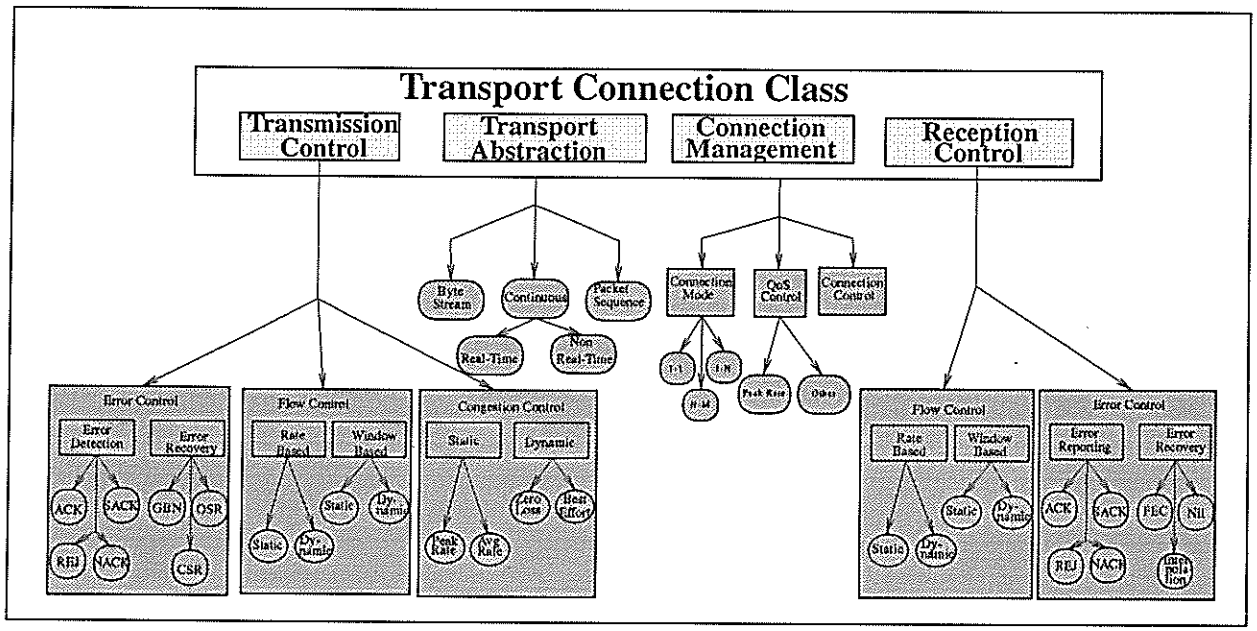


Figure 7: Transport Function Class Hierarchy Related Work

5.5. The Association and Presentation System (APS)

The way in which the transport requirements of an application's data stream are translated by the TFCS into a transport connection with suitable mechanisms and parameter values has been described above. Several such connections are carried by separate virtual circuits that can provide the bandwidth and delay requirements negotiated for each connection by the transport subsystem. The network in turn supports several access modes such as the call to carry these related virtual circuits used by an application. Since these connections are related by virtue of supporting a single application activity, there must be a mechanism to

exercise overall control over the communication activity. This is supported by the APS which provides the *association* mechanism[21] that binds the endpoints of an application and their connections. The APS has two main components, namely session management and transfer management. Session management creates the association given a list of endpoint addresses and performs operations such as adding and deleting endpoints, controlling attributes of the association such as accessibility, monitoring and modifiability. Accessibility controls how other endpoints join the association. Monitoring allows an endpoint to be notified of changes in the association attributes.

Transfer management allows an endpoint to setup connections and to specify their QoS. It then allows the creator to negotiate with the transport subsystem to satisfy the QoS requested. Another important function provided is enforcement of temporal relations between data on related connections. It is expected that

for continuous and isochronous data, the application will require some sort of temporal ordering to be maintained among the data samples of the streams. A well-known example is the "lip-syncing" problem in video communication.

6. Related Work in Multimedia Computing Systems

We briefly mention some of the work in multimedia computer systems. Pandora's box [22] aims to provide video support within the system with external hardware for a video handling subsystem and a pixel switch. The approach is specific to a single media type and is loosely integrated into the system. The ANSA group [6] has a more general approach wherein a case is made for real-time operating system support in order to provide flexibility. A distributed, object-oriented micro-kernel is advocated to provide system wide support. The IDCM [1] approach raises many difficult system software design issues in providing complete integration of continuous media, but does not address hardware architectural issues for achieving this. The ViewStation program[38] is investigating the design and deployment of distributed video systems. A software intensive approach and a kernel providing sufficient real-time support is described. I/O interconnection architectures are dealt with in [35]. A survey of scheduling algorithms that are suitable for multimedia systems is dealt with in [23].

References

[1] Anderson, D. P., Govindan, R., Homsy, G., Wahbe, R., "Integrated Digital Continuous Media: A Framework Based on MACH, X11, and TCP/IP," Technical Report, Department of EECS, University of California, Berkeley, August 1990.

[2] Arango, M. et. al, "Touring Machine: A Software Platform for Distributed Multimedia Applications," *IFIP*, Vancouver, May 1992.

[3] Arango, M. et. al, "Touring Machine: A Software Infrastructure to Support Multimedia Communications," *4th IEEE COMSOC, International Workshop on Multimedia*, Monterrey, April 1992.

[4] Arango, M. et. al, "The Application Programming Interface to the Touring Machine," Bellcore, Morristown NJ, 1992.

[5] Blakowski, G., "Supporting the Distributed Processing of Multimedia Information in an Object-Oriented, Heterogeneous Environment," *First International Workshop on Network and Operating System Support for Digital Audio and Video*, ICSI, Berkeley, November 1990.

[6] Blair, G. S., Coulson, G., Davies, N., Willaims, N., "The Role of Operating Systems in Object-Oriented Distributed Multimedia Platforms," Distributed Multimedia Research Group, Department of Computing, Lancaster University,1992.

[7] Bubenick, R., Gaddis, M.E. and DeHart, J.D., "Communicating with Virtual Paths and Virtual Channels," *Proceedings of the IEEE INFOCOM'92 Conference*, Florence, Italy, May 6-8, 1992.

[8] Cheriton, D.R., "The V Distributed System," *Communications of the ACM*, Vol. 31, No. 3, March 1988.

[9] Chesson, G., "XTP/PE Design Considerations," *Proceedings of the 1st International Workshop on High Speed Networks*, May 1989.

[10] Clark, D.D., "Modularity and Efficiency in Protocol Implementation," MIT Laboratory for Computer Science, RFC 817, July 1982.

[11] Cox, J. R. and Turner, J., "Project Zeus: Design of a Broadband Network and its Application on a University Campus," Washington University, Department of Computer Science, Technical Report WUCS-91-45, 1991.

[12] Crowcroft, J. and Wakeman, I., "A Technique for the Transmission of IP Datagrams Over B-ISDN Networks," Draft RFC, IETF, April 1992.

[13] Crutcher, L. and Grinham, J., "Connection Management for an Integrated-services Network and its Application to the Multi-media Communications of a Distributed Team," *First International Workshop on Network and Operating System Support for Digital Audio and Video*, ICSI, Berkeley, November 1990.

[14] DeHart, J.D., Gaddis, M.E. and Bubenik, R., "Connection Management Access Protocol (CMAP) Specification," Washington University, Department of Computer Science, Technical Report WUCS-92-01, February 1992.

[15] Doeringer et. al, "A Survey of Light weight Transport Protocols for High Speed Networks," *IEEE Transactions in Communication*, Vol. 38, No. 11, November 1990.

[16] Doshi, B.T. and Johri, P.K., "Communication Protocols For High Speed Packet Networks," *Computer Networks and ISDN Systems*, Vol. 24, 1992.

[17] Ferrari, D., Verma, D. C., "A Scheme for Real-time Channel Establishment for Wide-Area Networks,", IEEE Journal on Selected Areas in Communications, Vol. 8, pp. 368-379, April 1990.

[18] Gaddis, M.E, Bubenick, R. and DeHart, J.D., "A Call Model for Multipoint Communication in Switched Networks," *Proceedings of ICC'92*, Boston, June 1992.

[19] Gopal, G., Herman, G. and Vecchi, M.P., "The Touring Machine Project: Toward a Public Network Platform for Multimedia Applications," *Proceedings of SETSS*, Florence, March 1992.

[20] Hanko, J. et. al "Workstation Support for Time-Critical Applications," *2nd International Workshop*

on Support for Digital Audio and Video, Heidelberg, November 1991.

[21] Hong, Z. and McCoy, W., "An Associated Object Model for Distributed Systems," *Operating Systems Review*, October 1990.

[22] Hopper A., "Pandora - An Experimental System for Multimedia Applications," *Operating Systems Review*, April 1990.

[23] Herrtwich, R. G., "An Introduction to Real-Time Scheduling", Technical Report TR-90-035, Dept. of EECS, University of California at Berkeley, July 30, 1990.

[24] Hutchinson, N.C. and Peterson, L.L., "The x-kernel: An Architecture for Implementing Network Protocols,", *IEEE Transactions on Software Engineering*, January 1991.

[25] Kalfa Winfried, "Proposal of an External Processor Scheduling in Micro-Kernel based Operating Systems," Technical Report TR-92-028, Dept. of EECS, University of California, Berkeley, May, 1992.

[26] Leffler, S., et. al, *The Design and Implementation of the 4.3BSD UNIX Operating System*, Addison-Wesley, 1989.

[27] Leung, W.H. et. al, "A Software Architecture for Workstations Supporting Multimedia Conferencing in Packet Switching Networks," *IEEE Journal on Selected Areas in Communications*, Vol. 8, pp. 380-390, April 1990.

[28] Lippman, S.B., *The C++ Primer*, Addison-Wesley, 1991.

[29] Little, T.D.C. and Ghafoor, A., "Multimedia Synchronization Protocols," *IEEE Journal on Selected Areas in Communications*, Vol. 9, pp. 1368-1382, December 1991.

[30] Minzer, S., "Broadband ISDN and Asynchronous Transfer Mode (ATM)," *IEEE Communications Magazine*, Vol. 27, no. 9, pp. 17-24, September 1989.

[31] Netravali, A.N., Roome, W.D. and Sabnani, K.K., "Design and Implementation of a High Speed Protocol,." *IEEE Transactions on Communications*, Vol. 38, pp. 2010-2024, 1990.

[32] Nicolaou, C., "An Architecture for Real-Time Multimedia Communications Systems," *IEEE Journal on Selected Areas in Communications*, vol.8, no. 3, pp. 391-400, April 1990.

[33] Raman G. and Bovopoulos, A. D., "Design of a Multimedia Applications Development System," Technical Report WUCS-92-27, Department of Computer Science, Washington University, 1992.

[34] Ravindran, K., "Real-Time Synchronization of Multimedia Data Streams in High Speed Networks," Technical Report, Kansas State University, January 1992.

[35] Sah, A., Verma, D. C. and Oklobdjiza, V. G., "A Study of I/O Architecture for High Performance Next Generation Computers," Technical Report TR-91-008, ICSI, University of Berkeley, CA, 1991.

[36] Schill, A., "Objects and Distribution: Advantages, Problems and Solutions," *Intelligent Tools Conference*, Paris, November 1989.

[37] Schmidt, D.C., Box, D.F. and Suda, T., "ADAPTIVE A Flexible and Adaptive Transport System Architecture to Support Multimedia Applications on High-Speed Networks," Technical Report 92-46, Department of Information and Computer Science, University of California, Irvine, 1992.

[38] Tennenhouse, D., Ciholas, M. and Davin, J., "Telemedia, Networks and Systems," Group Annual Report (AR-001), July 1991-June 1992, Laboratory for Computer Science, MIT.

[39] Yavatkar, R., "Communication Support for Collaborative Multimedia Applications," Technical Report 181-91, Department of Computer Science, University of Kentucky, 1991.